Soft Error Protection via Fault-Resilient Data Representations

Muhammad M. Latif, Ravi Ramaseshan, Frank Mueller

North Carolina State University

Center for Embedded Systems Research

NC STATE UNIVERSITY
Department of Computer Science

CESR

Introduction

- Soft errors: Potential problem for tomorrow's processors
  - Exposure to radiation,
  - Lower signal to noise ratios.
- Approaches to fault tolerance:
  - Purely hardware – ECC memories
  - Purely software – EDDI
  - Hybrid: our approach.
- Our work focuses on:
  - Fault resilient data representations,
  - Programming constructs.
  - SEU (Single Event Upset) soft errors.

Related Work

- Error Detection by Duplicated Instructions (EDDI):
  - N. Oh et al, Stanford.
  - Pure software solution.
    - Redundant secondary data flow,
    - Control flow checking using basic block signatures.
  - Consistency checks at writes and branches.
- SWIFT [CGO2005]:
  - Reis et al, Princeton.
  - Assumption: Protected lower memory hierarchy.
  - Duplication only at register level
  - Consistency checks only on writes.

Objective and Challenges

- Identify fault-detecting data representation
  - Identify naturally fault tolerant programming constructs.
  - Constraints on the constructs patterns.
  - Automation through compiler transformations.

Contributions

- Fault resilient programming constructs and data representations.
  - Switch cases
  - Loop skewing
  - Value cloning
  - Parameter passing
  - Pointer traversals

Fault Resilient Switch Cases

- Idea: HammingDistance (case values) ≥ 2.
- Transformation: Enumerated data types.
- Benefit:
  - Avoid error checks with shadow variable,
  - Simplify control flow.

```
enum {A=0,B=1,C=2,D=3};
switch (x) {
  case A: if (x!=x')
    error(); ...
  case B: if (x!=x')
    error(); ...
  ...
}
```

```
enum {A=1,B=2,C=4,D=8};
switch (x) {
  case A: ...
  case B: ...
  case C: ...
  case D: ...
  default: error();
}
```

Fault Resilient Loop Skewing

- Idea: HammingDistance (iterator values) ≥ 2.
- Assumptions: Machine word size N loop upper bound < N.
- Benefit: Eliminate use of shadow variable.
- Transformation:
  - Skew loop induction variable and loop bounds.
  - Break long loops into factors of fault tolerant loops.
  - Unskew references of the original induction variable.

```
for (i=0, i'=0; i<10; i++, i'++) {
    if (i != i')
      error();
    ...
}
```

```
for (i=1; i<2^10; i=i«1) {
if (checksum(i)!=1)
  error();
  ...
}
```

Fault Resilient Value Cloning

- Idea: Pack multiple small integers.
- Assumptions:
  - sizeof (integer variable) = k * sizeof (machine word)
  - no overflow between packed variables.
- Benefit:
  - eliminate duplicated variables.
  - original and shadow instructions execute in parallel.

```
short int x = x'= 0xFF00,
    y = y'= ADEF,  z, z';
z  = x  ^ y
z' = x' ^ y'
if (z != z')
  error();
```

```
long int x = 0xFF00FF00,
  y = ADEFADEF, z;
z = x ^ y;
if (16MSB(z)!=16LSB(z))
  error();
```

Fault Resilient Parameter Passing

- Idea: Hash shadow parameters into a one parameter.
- Constraints: Can be applied on in-parameters.
- Transformation:
  - Xor all in-parameters into one.
  - Check parameters with Xor-ed parameter.
- Benefit: Reduce register pressure — avoid spills.

```
int foo(int a, int b, int c,
        int a', int b', int c') {
  int x, x';
  x = a + b + c;
  x' = a'+ b'+ c';
    if (x != x')
      error();
}
```

```
int foo(int a, int b, int c,
        int XOR_abc) {
  int x, x';
  x = a + b + c;
  x' = a + b + c;
    if(x'^XOR_abc != x^a^b^c)
      error();
}
```

Fault Resilient Pointer Traversal

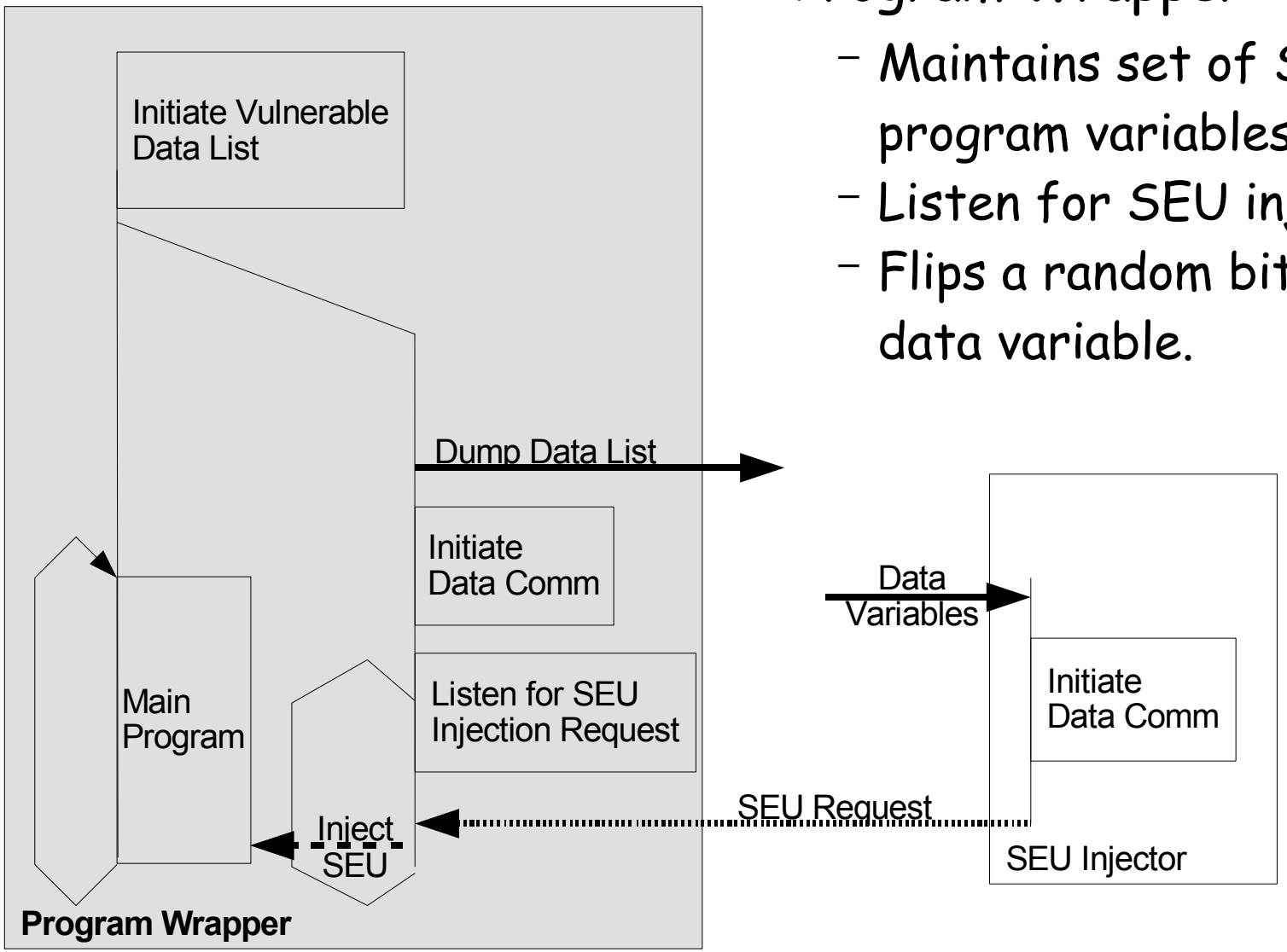
- Idea: HammingDistance (k * p) > 1
  - k = 1, 2, 3, ... and $p \neq 2^m$.
- Transformation:
  - Add padding so that sizeof (structure) $\neq 2^k$.
  - sizeof (structure) = p * sizeof (word size) for $p \neq 2^k$.
- Benefit: Avoid the use of shadow variable p'.
- Assumption: Hardware support to perform check.

```
struct st {char a, b};
st A[100], *p, *p';
for (p=A, p'=A; p<&A[99]; p++, p'++) {
  if (p != p')
    error ();
}
```

```
struct st {char a, b, pad};
st A[100], *p;
for(p=A; p<&A[99]; p++) {
  if ((p-A) % sizeof(st))
    error ();
}
```

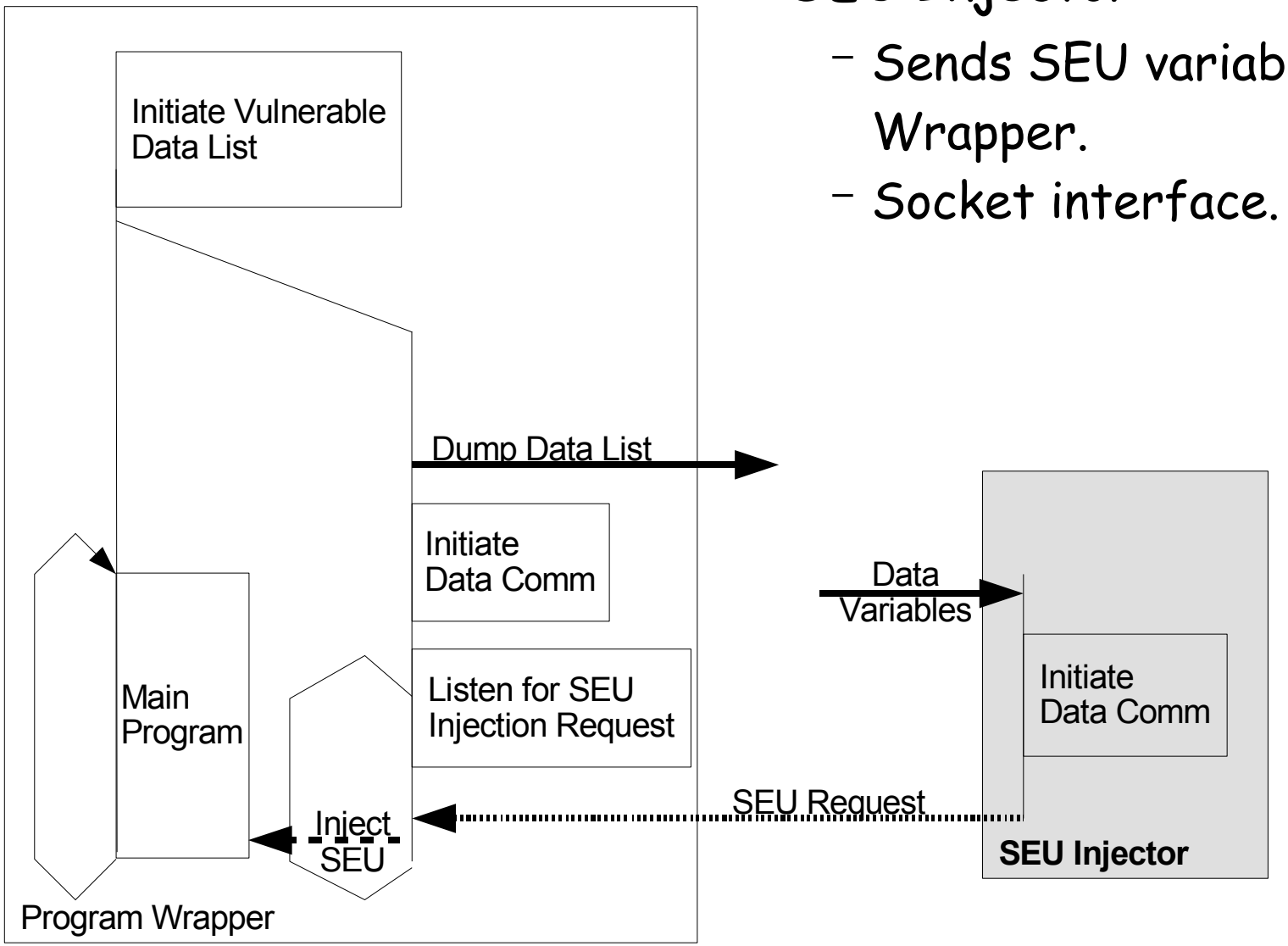
Experimental Framework

- Program Wrapper
  - Maintains set of SEU susceptible program variables.
  - Listen for SEU injection request.
  - Flips a random bit on requested data variable.



Experimental Framework

- SEU Injector
  - Sends SEU variable to Program Wrapper.
  - Socket interface.



Preliminary Results: Individual Transforms

Runtime Overhead (Percentage increase over base case)					
Construct	Switch	Loop	Pointer	Clone	Parameter
EDDI	7.27%	82.60%	8.14%	56.60%	100.60%
Resilient	4.46%	52.90%	1.19%	26.70%	64.30%

Code Size Overhead (Percentage increase over base case)					
Construct	Switch	Loop	Pointer	Clone	Parameter
EDDI	15.80%	18.20%	18.90%	19.10%	51.58%
Resilient	5.30%	11.70%	18.90%	15.60%	21.16%

- Limit study with micro benchmarks.
  - Manually transformed.
  - Base case: Benchmarks without any transformations.
  - Compiled with O0 to turn off optimizations.
- Average reduction in runtime overhead: 21.13%
- Average reduction in code size overhead: 10.18%

Preliminary Results: Combined Transforms

Bubble Sort Benchmark				
	EDDI		Resilient	
Optimization	O0	O1	O0	O1
Time (cycles)	12689	9432	10945	7044

- Bubble sort micro benchmark
  - Manually apply all proposed transformations
  - Skewing, pointer traversal, cloning, parameter passing
- Performance benefit over EDDI:
  - O0: 13.74% and O1: 25.32% lower overhead.

Conclusions and Future Work

- Programming constructs can be used for fault resilience.
- Performance benefit achieved using constructs:
  - Runtime overhead reduction: 2% to 36%.
  - Code size reduction up to 30%.
- Future Work
  - Automate transformation in OpenImpact compiler.
    - VLIW packing masks performance benefit.
  - Analyze coverage on large/standard benchmarks.